Introduction to Computer Organization – Fall 2023

Lab 4

Due: @11:55 PM, Wed September 27th

The following assignment is intended to be completed during your assigned lab period. One member of your group must submit the assignment to Gradescope by the posted deadline and indicate your group members when submitting the assignment. **Each group member must be present during the scheduled lab period in order to receive credit.**

Group names and uniqnames

Pod Member Name	**Uniqname**
Yuzhen Chen	yuzhench
Yuhan Zhang	zyuhan
Therron Montgomery	therronm

For each of the following problems, one person should act as the "scribe" and log the discussions of the group. You should rotate who is the scribe for each problem and indicate in the given space.

Problem 1: Short Answer *[8 points]*

Scribe: **[Scribe's name here]**

1. While waiting in office hours, you overhear your peer mention that, while they can work through caller-callee save register problems in the homework, they don't understand the motivation in the first place. In your own words (about 30 words or fewer), explain why saving/restoring register values with function calls is necessary, and what would happen if we didn't do it.
   Answer:
   The reason why saving/restoring register values with function calls is necessary is because, when we have nested functions, the inner function may use the same register as the outer function to store and value, if you don’t save them, the useful data will be overwritten.

2. The student thanks you for engaging in such wonderful peer instruction. They follow up asking why we do a mixture of caller-callee in practice. Is it a correctness issue? If not, what are we hoping to achieve with a mixture? (Again, about 30 words or fewer)
   Answer:

The reason why we mix caller-callee is to achieve better performance, reducing load instructions. It’s not a correctness problem, but it will affect the difficulty for debugging.

Problem 2: Reading LC2K *[15 Points]*

Scribe: **yuzhen chen**

Consider the function Find() and which has a few comments but limited documentation. You know that the function takes two arguments. The first is a pointer to an array (passed in r1) and the second is the number of elements in the array (passed in r2). The return value is found in r3. Answer the questions on the following page. You may use the LC2K simulator.

```
        lw      0       1       ArrayS
        lw      0       2       Count
        lw      0       3       Fcall
        jalr    3       7
        halt
Find    lw      0       6       NegOne  // r6=-1
        add     0       2       3       // r3=Count
Top     add     2       6       2       // Decrement Count
        add     1       2       4       // 4 is address
        lw      4       5       0       // ld array element
        beq     5       0       skip    // is  element 0?
        add     3       6       3       // if not sub 1
skip    beq     0       2       Done    // if Count=0 we are done
        beq     0       0       Top     // next iteration
Done    jalr    7       5               // return
Zero    .fill   0
NegOne  .fill   -1
Fcall   .fill   Find
ArrayS  .fill   Array
Count   .fill   4
Array   .fill   0
        .fill   1
        .fill   1111
        .fill   0
```

1) How many times will the line with the label “Find” get executed during this program? **[3]**

   Answer: 1 time

2) What is the value of r7 when the program halts? **[1]** **____4___**

3) What is the value of r3 when the program halts? **[1]** **____2___**

4) *In 10 words or fewer,* describe what the return value of the function is in terms of the array. Your answer should be something like “returns the maximum value in the array”. **[5]**
   Return the number of zero elements in the array

5) Someone suggests the code is redundant, and that the first lw instruction could replace the label ArrayS with Array. In 20 words or fewer, explain why this would not work, and what would happen if we made this change. **[5]**
   Answer:
   This loads the first array elements 0 to r1 instead of the array address. So the array access will be incorrect.

Problem 3: Link Like You've Never Linked Before! *[12 Points]*

Scribe: **[Scribe's name here]**

Read the following two files and fill in the rest of the symbol and relocation tables for each (on the next page). **Note that not all entry spaces in the tables provided need to be used.** We have completed one entry in each symbol and relocation table to show the format of each entry. Hint: see the Symbol and Relocation table guide on the website.

main.c

```
#include <string.h>
#include <stdlib.h>
#include <dog.h>

#define DOG_CNT 500 // Hint: see link

int tricks = 0;
int pet();
extern int bark(int dog);
extern int total_barks;
extern int[500] dog_happiness;
extern char* dog_str;

int main(){
   for(int i = 0; i<DOG_CNT; i++){
      dog_happiness[i] = 1;
   }
   char out [50];
   if(pet() > 50){
      strncpy(out, dog_str, 50);
   }
}

int pet(){
   static int petted_dogs = 0;
   for(int i = 0; i<DOG_CNT; i++){
      dog_happiness[i] *= 2;
      petted_dogs += 1;
      if (rand() & 1) {
         bark(i);
      }
   }
   return total_barks;
}
```

dog.c

```
1  #include <stdlib.h>
2
3  extern int tricks;
4  int[500] dog_happiness;
5  int total_barks = 10;
6  int flag;
7
8  void give_treat(int happiness);
9
10 int bark(int dog){
11    total_barks += 1;
12    if (rand() & 1) {
13       give_treat(dog);
14    }
15    return flag;
16 }
17
18 void give_treat(int happiness){
19    if (happiness && flag) {
20       tricks++;
21    }
22 }
23
24
25
26
27
28
29
30
31
32
33
34
35
```

Problem 3: Link Like You've Never Linked Before! *[12 Points]*

Scribe: **[Scribe's name here]**

Read the following two files and fill in the rest of the symbol and relocation tables for each (on the next page). **Note that not all entry spaces in the tables provided need to be used.** We have completed one entry in each symbol and relocation table to show the format of each entry. Hint: see the Symbol and Relocation table guide on the website.

main.c

```
#include <string.h>
#include <stdlib.h>
#include <dog.h>

#define DOG_CNT 500 // Hint: see link

int tricks = 0;
int pet();
extern int bark(int dog);
extern int total_barks;
extern int[500] dog_happiness;
extern char* dog_str;

int main(){
   for(int i = 0; i<DOG_CNT; i++){
      dog_happiness[i] = 1;
   }
   char out [50];
   if(pet() > 50){
      strncpy(out, dog_str, 50);
   }
}

int pet(){
   static int petted_dogs = 0;
   for(int i = 0; i<DOG_CNT; i++){
      dog_happiness[i] *= 2;
      petted_dogs += 1;

      if (rand() & 1) {
        bark(i);
      }
   }
  return total_barks;
}
```

dog.c

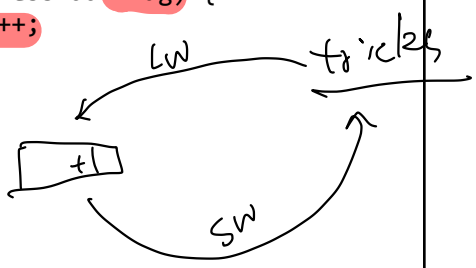
```
#include <stdlib.h>

extern int tricks;
int[500] dog_happiness;
int total_barks = 10;
int flag;

void give_treat(int happiness);

int bark(int dog){
   total_barks += 1;
   if (rand() & 1) {
      give_treat(dog);
   }
   return flag;
}

void give_treat(int happiness){
   if (happiness && flag) {
      tricks++;
   }
}
```



Problem 3: Link Like You've Never Linked Before! *[12 Points]*

Scribe: **[Scribe's name here]**

Read the following two files and fill in the rest of the symbol and relocation tables for each (on the next page). **Note that not all entry spaces in the tables provided need to be used.** We have completed one entry in each symbol and relocation table to show the format of each entry. Hint: see the Symbol and Relocation table guide on the website.

main.c

```
#include <string.h>
#include <stdlib.h>
#include <dog.h>

#define DOG_CNT 500 // Hint: see link

int tricks = 0;
int pet();
extern int bark(int dog);
extern int total_barks;
extern int[500] dog_happiness;
extern char* dog_str;

int main(){
   for(int i = 0; i<DOG_CNT; i++){
      dog_happiness[i] = 1;
   }
   char out [50];
   if(pet() > 50){
      strncpy(out, dog_str, 50);
   }
}

int pet(){
   static int petted_dogs = 0;
   for(int i = 0; i<DOG_CNT; i++){
      dog_happiness[i] *= 2;
      petted_dogs += 1;

      if (rand() & 1) {
        bark(i);
      }
   }
   return total_barks;
}
```

dog.c

```
1  #include <stdlib.h>
2
3  extern int tricks;
4  int[500] dog_happiness;
5  int total_barks = 10;
6  int flag;
7
8  void give_treat(int happiness);
9
10 int bark(int dog){
11    total_barks += 1;
12    if (rand() & 1) {
13       give_treat(dog);
14    }
15    return flag;
16 }
17
18 void give_treat(int happiness){
19    if (happiness && flag) {
20       tricks++;
21    }
22 }
23
24
25
26
27
28
29
30
31
32
33
34
35
```

global variable.
↓ global function.

main.o symbol table		dog.o symbol table	
Symbol	**Type (T/D/U)**	**Symbol**	**Type (T/D/U)**
tricks	D	tricks	U
pet()	T	dog_happiness	D
bark()	U	total_barks	D
total_barks	U	flag	U
dog_happiness	U	give_treat()	T
dog_str	U	bark()	T
main()	T	rand()	U
Strncpy	U		
petted_dogs	D		
rand()	U		

main.o relocation table			dog.o relocation table		
Line	**Instruction (LD, ST, BL)**	**Symbol**	**Line**	**Instruction (LD, ST, BL)**	**Symbol**
16	STUR	dog_happiness	11	STUR	total_barks
20	BL	Strncpy()	11	LD	total_barks
20	LD	dog_str	12	BL	rand()
24	BL	pet()	15	LD	flag
27	LD	dog_happiness	19	LD	flag
27	ST	dog_happiness	20	LD	tricks
28	LD	petted_dogs	20	ST	tricks.
28	ST	petted_dogs			
30	BL	rand()			
31	BL	bark()			
34	LD	total_barks			

main.o symbol table		***dog.o symbol table***	
Symbol	***Type (T/D/U)***	***Symbol***	***Type (T/D/U)***
tricks	*D*	*tricks*	*U*

main.o relocation table			***dog.o relocation table***		
Line	***Instruction (LD, ST, BL)***	***Symbol***	***Line***	***Instruction (LD, ST, BL)***	***Symbol***
16	*STUR*	*dog_happiness*	*11*	*STUR*	*total_barks*

Problem 4: The Best of the Tests *[15 points, Autograded]*

With project 2a right around the corner, let's get ahead and find some of those buggy instructor solutions!

- For this problem, your group will be submitting (to autograder.io) assembly code AND its correct output object file when run through the project 2a assembler.
- Once you have written test cases that expose the bugs, you must write the corresponding correct object file output and submit it to the autograder for full credit.
- For every bug that is caught, you will receive 5 points. So for full credit, you need to catch 3 bugs with your tests, and provide the correct output for each.
- In addition to the constraints listed in the project, each LC2K program you write must be limited to **4 lines** or fewer.
- No test cases should cause errors on a correct assembler.
- Each submission will be limited to 3 test cases, but fewer may be needed.
- Each output file name must be the same as the assembly, with the extension changed to .obj.
- You are free to resubmit tests you wrote for P1a, and you may submit these for P2a as well without honor code penalty (as long as they were written by members within your group or provided in course materials).
- You are encouraged to use (and submit) these test cases if you are still working on P2a. A great strategy is to run these test cases on your assembler and "diff" your output with the correct output any time you make a change.